

**UNITED STATES PATENT APPLICATION**

*of*

**Jacob M. McGuire**

*for a*

**HIGHLY, SCALABLE LEAST CONNECTIONS LOAD BALANCING**

# HIGHLY, SCALABLE LEAST CONNECTIONS LOAD BALANCING

## CROSS-REFERENCE TO RELATED APPLICATIONS

5 The present application claims priority from U.S. Provisional Patent Application  
Serial No. 60/236,555, which was filed on September 29, 2000, by Jacob M. McGuire for  
a HIGHLY, SCALABLE LEAST CONNECTIONS LOAD BALANCING and is hereby  
incorporated by reference.

## BACKGROUND OF THE INVENTION

### *Field of the Invention*

10 The present invention relates to load balancing traffic among a plurality of serv-  
ers, and in particular, to a least connections load balancing method.

### *Background Information*

15 Computer networks typically use file servers which frequently operate under a  
client-server paradigm. Under this model, multiple clients can make input/output (I/O)  
requests which are directed to a particular resource on the network. A server on the net-  
work receives and carries out the I/O requests. When the server receives multiple I/O  
requests, the server may choose to service them one at a time. I/O requests which are not  
being processed typically wait until the server is ready to receive more requests. As a  
result, the server can become a bottleneck in the network.

20 Typically, it is desirable to distribute various client requests among the plurality  
of servers. In these instances, it requires collaboration as to how the various client re-  
quests are to be distributed among those various servers. This may be performed through

load balancing. Server load balancing allows for a group of real servers (a server farm) to be represented as a single virtual server entity wherein the traffic is balanced among the plurality of servers. One method to obtain server load balancing is to use a round-robin method. With this method, a new connection between a client and a real server is performed by choosing the real servers in a circular manner wherein a connection is made if the chosen server has capacity to handle the connection. However, the round-robin method does not insure that the various real servers are indeed effectively load balanced.

Another method for server load balancing is a least connections method in which a new connection is assigned to the real server with the least number of currently active connections. Compared with the round-robin method, the least connections method provides for a more accurate load balancing of the servers; however, it is rather complex and consumes a fair amount of processing time of the device that is performing the load balancing. For instance, the method sends a new connection to a server which has the lowest metric, wherein the metric is defined as the number of connections on the server divided by the weight (or capacity) of the server. This metric is kept as a quotient/remainder pair. To keep track of the metric and the remainder, integer division is typically performed on all servers every time a connection is added or removed.

On a different note, it is desired to flexibly increase the number of real servers as demand increases for resources at the server farm. However, one aspect of this problem is that as greater numbers of real servers are added, the load balancing process slows down, which may require replacing the load balancing device with one that processes at a faster speed. For instance, consider a method in which the load balancing device sequentially tests a list of servers for least connections. One reason for the slowdown may be that as the list of potential servers for connections increase, more time is needed for the device to find the server with the lowest number of active connections.

## SUMMARY OF THE INVENTION

A load balancing device according to an embodiment of the invention uses a predictor that comprises a plurality of Least Connections Control Blocks (LCCBs) configured to keep track of the real servers with active connections. To speed up the search for

5 a real server with the least number of active connections, an LCCB is kept for each server metric. A metric is defined as the number of connections on a server divided by the weight (or capacity) of the server. This metric is kept as a quotient/remainder pair. The predictor sends out the real server address with the lowest metric from the LCCB with the least connections whenever a new connection is required by the load balancing device. According to the embodiment, a list of metric/pointer pairs is kept by the associated LCCB, one corresponding to each real server, and when a connection is requested, the real server with the lowest metric is used for the connection which is pointed to by the pointer of the metric/pointer pair.

10 Each of the LCCBs is associated with a list of server metric/pointers that points to servers having a similar number of connections. Because the LCCBs are kept in sorted order (in terms of the number of connections in a server), finding the least loaded server is simply a matter of selecting the LCCB with the lowest connections and taking the first real server with the lowest metric (which is pointed to by the LCCB). The metric of the server is revised and the server metric/pointer is transferred to another LCCB which corresponds to the server's new number of connections. In particular, when a connection is added to the server, the remainder (which is also called a "slope") is incremented, and when a connection is removed, the slope is decremented. When the slope equals the weight, or goes below zero, the metric is updated and the slope reset.

20 As the number of connections of a real server changes, the server metric is removed from its current LCCB. Because the predictor creates/destroys connections sequentially, the request that a connection be created or destroyed occurs at most one at a time. Thus, if the connection of the real server changes, at most, the server metric/pointer is moved to one of the immediate LCCBs. For example, if a server metric/pointer is on the list of the LCCB having two connections and another connection is added to it, the metric is revised and that server metric/pointer is moved to an adjacent LCCB which is associated with three connections. The server metric/pointer is transferred to the new LCCB via a "double linked" connector, wherein the new LCCB subsequently updates and sorts its server metric/pointer list. The new LCCB then points to the server with the lowest metric. Conversely, if a server in the LCCB with two connections loses a connection,

the metric is revised and the metric/pointer is moved to the adjacent LCCB which is associated with one connection. The LCCB transfers the server metric/pointer to the new LCCB via the double linked connector. The new LCCB then updates and sorts its server metric/pointer list and points to the server with the lowest metric.

## BRIEF DESCRIPTION OF THE DRAWINGS

The invention description below refers to the accompanying drawings, of which:  
Fig. 1 illustrates a network in which the present invention may be implemented;  
Fig. 2 illustrates a virtual server including a server farm and a load balancing device;

Fig. 3 is a schematic diagram of the load balancing device in which an embodiment of the invention may be implemented;

Fig. 4 is a schematic diagram of a processing unit of the load balancing device constructed in accordance with an embodiment of the invention;

Fig. 5a is a predictor constructed in accordance with an embodiment of the invention; and

Fig. 5b is the predictor of Fig. 5a in which a redundant Least Connections Control Block (LCCB) has been removed in accordance with an embodiment of the invention.

## DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

**Fig. 1** shows an illustrative network 100 including a network cloud 110 which may be a combination of transmission lines, backbones, switches, routers and repeaters, dependent on the extent of the network, and allows various devices connected to the network to communicate with each other. These various devices may include simple computers such as personal computers (PCs) and workstations 112, (herein referred to as clients) that transmit or receive data, or they may be sophisticated servers 114 that store various resources. The network 100 may also include virtual servers 200, which operate in accordance with an embodiment of the present invention, to exchange data with clients

112. The network cloud 110, for instance, may be the Internet that interconnects large numbers of these simple computers, servers and virtual servers to facilitate the exchange of information. Information, in one example, flows in packets having attached headers that include, among others, source and destination addresses that inform the devices encountered in the network as to how the packets are to be forwarded so that they may reach their destination.

**Fig. 2** is an exemplary virtual server 200 that includes a server farm 201 and a load balancing device 300 that provides for server load balancing and allows for clients to be represented by a group of network servers. From the clients' perspective, the virtual server 200 is a single server entity. One purpose of the virtual server is to facilitate scalability and to expedite resource availability for the clients. Moreover, the addition of new servers and/or the removal/failure of existing servers can occur at any time without affecting the resource availability provided by the virtual server. Server load balancing allows for traffic to be balanced among the servers and in certain instances, limit traffic to certain clients or servers. The servers that comprise the server farm 201 are herein referred to as "real servers" 202.

According to one embodiment, the clients 112 are configured to communicate with the virtual server 200 using an IP address that represents the virtual server (hereinafter referred to as "virtual IP address."). When a client 112 initiates a connection with the virtual server 200, this connection is received by a load balancing device 300 which, in turn, chooses a real server 202 from the server farm 201 using the novel load balancer (to be described herein). Although the load balancing device 300 is shown as being separate from the server farm 201, in another embodiment, it may be integrated into one of the servers 202 in the server farm 201.

**Fig. 3** is a diagram of an illustrative load balancing device 300, such as a switch, that directs traffic between the clients 112 and the real servers 202. The load balancing device comprises a processing unit 400, a forwarding engine 304 and a plurality of input/output (I/O) ports 340. The processing unit 400 may be a single processor or a plurality of processors that manage the load balancing device and execute, among other things, routing protocols. The processing unit 400 also executes a load balancing algo-

5 rithm according to one embodiment of the invention. The forwarding engine 304 is a switch component that performs packet switching of packets received from the I/O ports 340, some of which are connected to the network cloud 110 and others of which are connected to the server farm 201. The load balancing device 300, in one embodiment, performs packet redirection (in another embodiment, it may be a packet translation) to and from the real servers 202 in the server farm 201 on traffic from clients requested through the virtual IP address. The load balancing device 300 uses a unique real server address to address a specific real server.

10 Communication between a client and a server farm is established by a “connection.” A connection is a series of Internet Protocol (IP) packets associated with the same pair of IP addresses and the same Transport Control Protocol/User Datagram Protocol (TCP/UDP) ports. The information that describes packets belonging to a unique client/server connection and allows for a packet to be sent to and from a real server includes:

- 15 Source IP address;
- Destination IP address;
- Protocol;
- Source TCP/UDP port;
- Destination TCP/UDP port; and
- 20 IP address to forward the packet to.

The following example describes how a load-balanced TCP connection between a client and a server is processed by the load balancing device, which in this instance is a switch. The TCP protocol is connection-oriented and has known protocol messages for activation and deactivation of TCP sessions. Thus, under one method, a state machine is used to correlate TCP packets such as SYN, SYN/ACK, FIN and RST for realizing a connection is to be added/removed or for determining the number of connections per server. To expedite the load balancing process, an Access List Control 320 may be used to filter the TCP packets that indicate that a load balancing process is to take place.

Referring to **Fig. 4**, when the processing unit 400 receives the packet, the processor 402 sees that the index corresponds to load balancing, and processes the packet using a load balancer 410, which may be implemented in hardware or software. The load balancer 410 maps this index to a virtual server instance, decodes the packet and searches a connection database 412 to find a connection. If it fails to find a connection, the load balancer 410 chooses a real server (i.e., the server address) using a predictor 500 (which may be implemented in hardware or software) and creates a connection in the connection database 412. The predictor 500 will be further described herein. The chosen real server address is conveyed to the forwarding engine 304 as the SYN packet is placed on the bus 302. The forwarding engine 304 then "caches" the server address in the forwarding table 334 as the SYN packet is forwarded to the chosen real server.

7 H:\112\025\0460\PROSECUT\PATAPP2.doc 11/01/00 2:28 PM



redirects the packet to the processing unit 400 while marking it with a software index that indicates that it pertains to load balancing. Meanwhile, ASIC2 332 creates a flow instance in its forwarding table 334. Note that the forwarding unit 330 keeps two separate caches; one for the inbound packets and one for the outbound packets. Of course, this  
5 may be modified according to a desired result. The processor 402 in the processing unit 400 processes the packet using the load balancer 410 which maps this index to a virtual server instance, decodes the packet and finds the connection in the connection database 412. The client address is thus found and the packet is placed on the bus 302. Logic currently encoded on ASIC2 332, meanwhile, learns the client address for this flow and  
10 caches this in the forwarding table 334 as the SYN/ACK packet is forwarded to the client.

The client then sends a TCP ACK packet. When the switch 300 receives this packet, the ASIC1 322 does not find a match in the ACL 324 and permits the packet to be forwarded as opposed to redirecting it to the processing unit 400. Because the forward-  
15 ing unit 330 has a flow entry in its forwarding table 334 for this flow, the forwarding engine 304 sends the packet directly to the associated real server 202.

TCP Data packets are now exchanged between the client and the server during this TCP session. Because the flags of interest i.e., SYN, FIN, and RST are not set on any of these packets, and the forwarding unit 330 has the flow entries in its forwarding  
20 table 334, the forwarding engine 304 sends the packets directly to the server or client without consulting the processing unit 400 (i.e., the load balancer).

Towards the end of the session, the client sends a TCP RST packet. When the switch receives this packet, the ASIC1 322 consults the ACL 324 to determine if it needs to be redirected. If so, the ASIC1 322 redirects the RST packet to the processing unit 400  
25 while marking it with a software index that indicates that it pertains to load balancing. The processing unit 400 receives this packet and the processor 402 sees that the index corresponds to load balancing and maps this index to a virtual server instance, decodes the packet and finds the connection in the connection database 412. The connection is then removed from the connection database 412 and the memory space consumed by the  
30 connection is returned to the free connection pool.

When the client issues a UDP packet using the virtual IP address and the switch 300 receives this packet, ASIC1 322 permits the packet to be forwarded since no matching entry exists in the ACL 324. Because the forwarding unit 330 does not have a corresponding flow in its forwarding table 334, it causes the ASIC2 332 to create a new flow and to forward the packet to the processing unit 400 where a lookup operation is performed to locate the route. The processing unit 400 receives this packet and because there is no software index that indicates load balancing, the processor consults a Common Packet Filter (CPF) 404. The CPF 404 forwards this packet to the load balancer 410 because it has a matching and best-fit packet filter. The load balancer 410 maps this index to a virtual server instance, decodes the packet, searches the connection database 412, and fails to find a connection. The load balancer then uses a predictor 500 to choose a real server, creates a connection, inserts it into the connection database 412, and starts the connection's idle timer. The real server IP address is found and this is used forward the packet on the bus 302. The ASIC2 332 learns the server address and caches this in the forwarding table 334 as the packet is forwarded to the chosen real server 201.

9  
H:\112\025\0460\PROSECUT\PATAPP2.doc 11/01/00 2:28 PM

finds the connection in the connection database 412, and ascertains the client's IP address. This IP address is used to forward the packet on the bus 302. Meanwhile, the ASIC2 332 learns the client's IP address while the packet is forwarded to the client.

UDP Data packets are exchanged between the client and the server using the same set of ports. Since ASIC1 322 simple permits the packets and the forwarding unit 330  
5 has the flow entries in its forwarding table 334, the forwarding engine 304 sends the packets directly to the server or client. Eventually, the UDP connection will become idle for a period of time exceeding the idle timer. At this point, the load balancer 410 will timeout the connection.

10 **Fig. 5a** shows an exemplary predictor 500 constructed in accordance with an embodiment of the invention. The predictor 500 may be constructed using hardware (such as ASIC) or software. The predictor comprises a plurality of Least Connections Control Blocks (LCCBs) 502 that keeps track of real servers with active connections. To speed up the search for the real server with the least number of active connections, an LCCB is kept for each server metric. A metric is defined as the number of connections on a server  
15 divided by its weight (or capacity) of the server. This metric is kept as a quotient/remainder pair. The predictor 500 sends out via a selector 508 the real server address with the lowest metric from the LCCB with the least connections whenever a new connection is required by the load balancer 410. According to the embodiment, a list of metric/pointer pairs 504 is kept in a memory of the associated LCCB, one corresponding  
20 to each real server 202, and when a connection is requested, the real server with the lowest metric is used for the connection which is pointed to by the pointer of the metric/pointer pair.

Each of the LCCBs is associated with a list of server metric/pointers that points to  
25 servers having similar number of connections. Because the LCCBs are kept in a sorted order (in terms of the number of connections in a server), finding the least loaded server 202 is simply a matter of the selector 508 selecting the LCCB with the lowest connection and taking the first real server with the lowest metric (represented by the metric/pointer pair 504) which is pointed to by a pointer 506 of that LCCB. As the real server address is  
30 supplied to the load balancer, the metric of the server is revised and the server met-

5

10

25

5

10

15

20